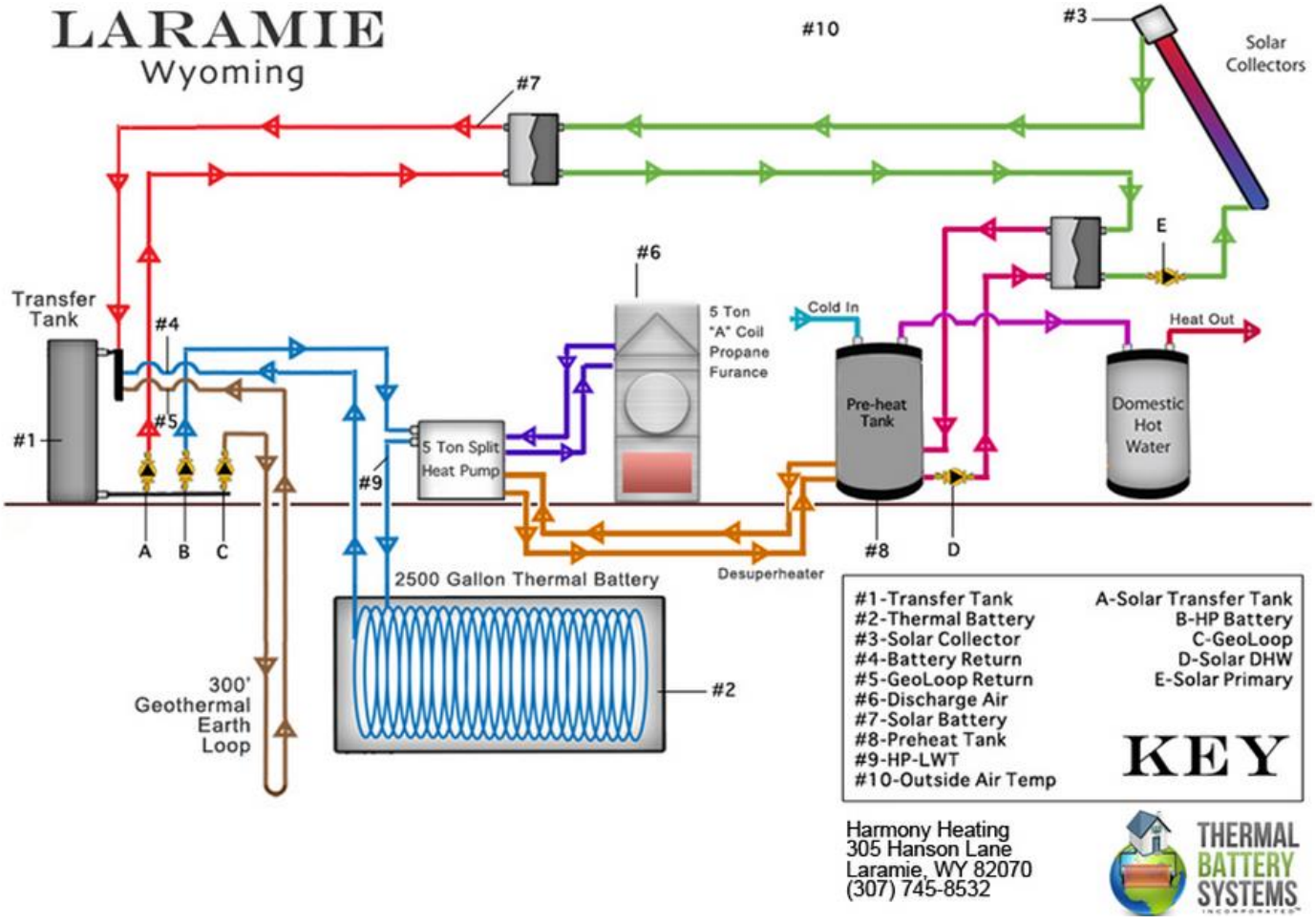
Integrated Solar Thermal - 2 Stage GSHP - Desuperheater - 3rd Stage Propane Furnace System

Plumbing Schematic & eZEio Integrated Controls

(Residential 2 Pipe Configuration)



Circulatory Pump Loops

A = Solar Transfer - solar thermal heat exchanger #1 loop to transfer tank loop
B = **GSHP Battery** – transfer tank to GSHP to 2500 gal. thermal battery loop
C = **Geothermal Earth Loop** – 300’ geothermal earth loop to transfer tank loop
D = **Solar Domestic Hot Water (DHW)** – preheat tank to solar thermal heat exchanger #2 loop
E = **Solar Primary** - solar thermal heat exchanger #2 loop to solar collector

Temperature & Humidity Sensors

T1 = Transfer Tank (GSHP EWT)
T2 = Thermal Battery
T3 = Solar Collector
T4 = Thermal Battery Return
T5 = Geothermal Loop Return
T6 = Discharge Air
T7 = Solar Thermal Battery
T8 = Preheat Tank for Domestic Hot Water (DHW)
T9 = GSHP LWT (leaving water temperature)
T10 = Outside Air (reset)
T11 = Inside Air & Humidity - Zone 1 (feedback)
T12 = Inside Air & Humidity - Zone 2 (feedback)
T13 = Inside Air & Humidity - Zone 3 (feedback)
Y1 = 1st Stage GSHP Compressor Terminal
Y2 = 2nd Stage GSHP Compressor Terminal

Most HVAC, solar thermal, solar photovoltaic (PV), domestic hot water (DHW), and energy recovery ventilators (ERVs) have individual controls that are not adequate for integrated systems like that described above. A single device that can provide integrated controls for multiple energy resources through custom hardware and software development is appealing. This is particularly true if such an integrated device can provide valuable data logging, and time saving diagnostics-troubleshooting tools for HVAC technicians and third party monitoring including state-of-the-art analytics.

In terms of energy efficiency, the Oak Ridge National Lab and leading control manufacturers such as tekmar reveal that integration of GSHPs, hydronic/radiant floor heating, thermal batteries (including high thermal mass ICF/concrete structures), cooling, and DHW with a second stage fan-coil/air-handler can reduce HVAC energy consumption by up to 50% in comparison with forced air alone. When programmable features such as Outside Temperature Reset, Indoor Temperature Feedback, Dew Point Reset, Strategic Zoning, Zoning Synchronization, Data Logging, Basic Analytics, and Smart Controls are programmed into the eZEio Controller and integrated with Real-time Monitoring, Warning Notifications, and state-of-the-art Maintenance & Diagnostic Systems, this can result in unprecedented energy efficiencies.

The eZEio™ Integrated Controller

The eZEio™ Integrated Controller is a general purpose Input/Output device, capable of monitoring, logging and controlling a wide range of devices and equipment through industry standard connections over the Internet.

The eZEio™ Integrated Controller automatically establishes a secure link to an array of redundant and secure servers, allowing live access from a standard web browser and eliminating the need for fixed IP, complex firewall setup or special software.

Multiple eEZio™ Controllers can be set up under a single user account, thus providing a simple

overview of the status of any number of sites spread out geographically from anywhere in the world.

All configuration, control, and access is accomplished by logging into the secure servers via the Internet at **www.ezecontrol.com**.

The eZEio™ controller is available in the following configurations:

Model	Wireless sensors	GSM/3G/GPS
ezeio	-	-
ezeio-W	YES	-
ezeio-G	-	YES
ezeio-GW	YES	YES

Base Model

The eZEio connects to the Internet via standard 10/100 Ethernet and uses wired peripherals via Modbus, MicroLAN and discrete in/outputs.

Wireless Sensors

The eZEio can be configured with a wireless transceiver module to allow for wireless sensors and expansion units. The wireless protocol is encrypted and only wireless devices from eZE System can communicate with the eZEio controller over this network. Typical indoor range is about 30m (100ft) but that obviously depends on wall materials and other environmental factors.

GSM/3G/GPS

When configured with a built-in GSM modem, the eZEio can communicate with the Internet via cell service. This requires service from a local cell provider and only GSM systems are supported. The controller will use the physical Ethernet path if it is available, but automatically switches to GSM if it can't communicate over Ethernet.

The GSM modem also supports GPS, so with the addition of an external antenna, the controller will have access to its position in real time. All versions run the same software and all other features are the same.

Some features depend on the service level. All versions come with four months of Basic Service, which allows for logging data from five inputs. See the company web page at www.ezesys.com for details about service levels and monthly cost for advanced services.

The eZEio Controller operates using the PAWN programming/scripting language to provide custom integrated controls for multiple sources of energy resources at a fraction of the price of purchasing individual controls that are often incompatible.

Custom Programming of eZEio Integrated Controls

Whatever the language or tool, programming is a craft that requires a particular approach to problem solving —or even a particular way of thinking about activities and information. It centers around analysis: “what needs to be done under which conditions and criteria”, reductionism: subdividing a large task into a hierarchy of smaller tasks, and synthesis: bringing it all together.

Before starting to build a program, one must understand the problem at hand. Once you have a good idea of what the program must do, you have to think about a solution in small discrete and deterministic steps. Each step may have inputs and outputs and you can describe the function of the steps in terms of the inputs and outputs alone—this is what the term discrete refers to.

Formulating Operational Controls

Conditions Affecting Circulation Pump A (CPA)

BOTH LINES IN SHOULDERS – 1ST LINE ONLY IN WINTER
If T3 > T1+5, then turn CPA ON until T3 < T1+1, then turn CPA OFF
If T8 < 90°F, then turn CPA OFF until T8 > 120°F, then turn CPA ON
If a conflict exists, OFF trumps ON

Run Away Conditions:
If T3 > 200°F, then turn CPA OFF
If T1 > 100°F, then turn CPA OFF

Fail Safe Conditions:
If communication is lost, FAIL OFF

Conditions Affecting Circulation Pump B (CPB)

If T1 > T2+5, then turn CPB ON until T1 < T2+1, then turn CPB OFF
If T2 > 70°F, then turn CPB ON until T2 < 66°F, then turn CPB OFF
If the compressor Y is ON, then turn CPB ON
If the compressor Y is OFF, then turn CPB OFF
If a conflict exists, ON trumps OFF

Fail Safe Conditions:
If communication is lost, FAIL ON

Conditions Affecting Circulation Pump C (CPC)

If T1 < 42°F, then turn CPC ON until T1 > 45°F, then turn CPC OFF
If T1 > 70°F, then turn CPC ON until T1 < 66°F, then turn CPC OFF

Fail Safe Conditions:
If communication is lost, FAIL ON

Conditions Affecting Circulation Pump D (CPD)

ON IN SUMMER AND SHOULDERS – OFF IN WINTER
If T3 > T8+5, then turn CPD ON until T3 = T8+1, then turn CPD OFF

Run Away Conditions:
If T3 > 200°F, then turn CPD OFF

Fail Safe Conditions:
If communication is lost, FAIL OFF

Conditions Affecting Circulation Pump E (CPE)

If either CPA or CPD are turned ON, then turn CPE ON
If CPA and CPD are both turned OFF, then turn CPE OFF

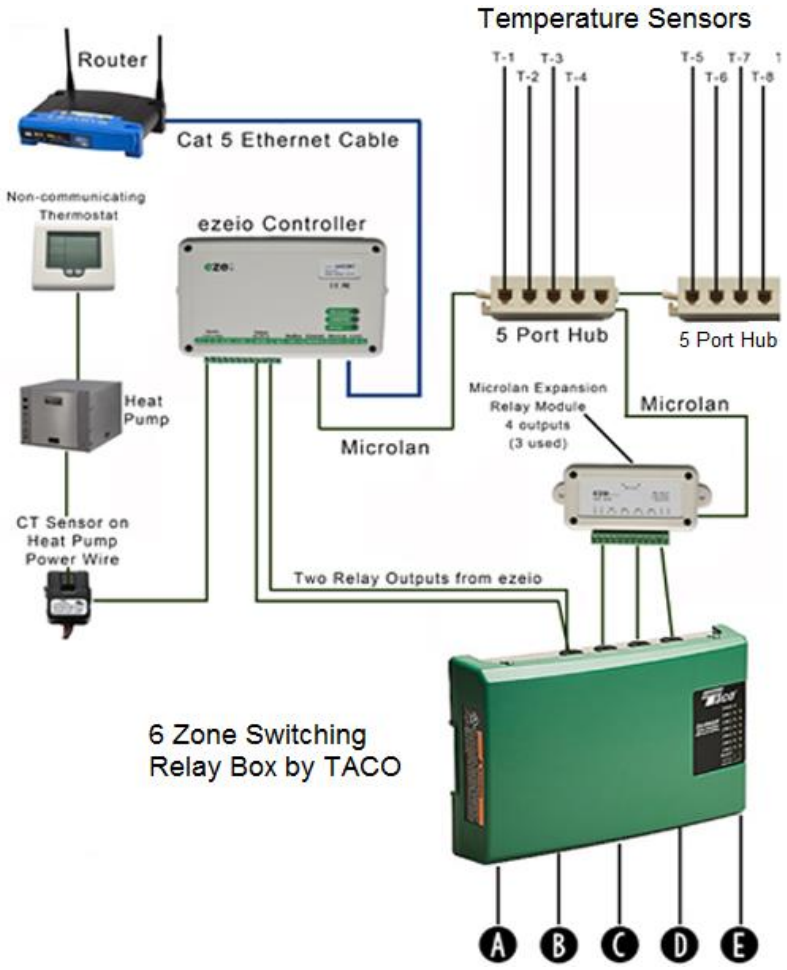
Fail Safe Conditions:
If communication is lost, FAIL OFF

As soon as you cannot explain a step without referring to other "steps" in the program, you have either a hidden input or an implied assumption, and you would do well to reconsider the analysis. The term deterministic is meant to say that the functioning of a step or a sequence of steps may not depend on (human) interpretation or judgment.

Many beginning programmers find it helpful to write down these steps in a flow chart (or some other schematic). During the analysis of the problem and its solution, you will often find that some steps are rather big —too big to be annotated in a little box in a flow chart. Such larger steps must be subdivided further, perhaps in another flow chart.

Basic Flow Chart/Wiring Schematic of eZEio Integrated Control System

[eZEio Controller, inputs and outputs, one wire (cat 5) connections, ModBus (both sensor and control inputs and outputs) MicroLAN (only sensor inputs and outputs) Firmware schematics, etc., for Integrated Wiring of Controls and System Components including Data Logging, Basic and Advanced Web-based Monitoring, Warning Notifications, & Diagnostic Tools]



Control Parameters

Conditions Affecting Pump A:
BOTH LINES IN SHOULDERS- 1st LINE ONLY IN WINTER
If T3 > T1 + 5, then turn A ON until T3 < T1 + 1, then turn A OFF
If T8 < 90°F then turn A OFF until T8 > 120°F then turn A ON
If a conflict exists, OFF trumps ON
Run Away Conditions:
If T3 > 200°F, then turn A OFF
If T1 > 100°F, then turn A OFF
Fail Safe Conditions:
If communication is lost, FAIL OFF

Conditions Affecting Pump B:
If T1 > T2 + 5, then turn B ON until T1 < T2 + 1, then turn B OFF
If T2 > 70°F then turn B ON until T2 < 66°F, then turn B OFF
If the compressor Y is ON, then turn B ON
It the compressor Y is OFF, then turn B OFF
If a conflict exists, ON trumps OFF
Fail Safe Conditions:
If communication is lost, FAIL ON

Conditions Affecting Pump C:
If T1 < 42°F, then turn C ON until T > 45°F then turn C OFF
If T1 > 70°F, then turn C ON until T < 66°F then turn C OFF
Fail Safe Conditions:
If communication is lost, FAIL ON

Conditions Affecting Pump D:
Runs in summer and shoulders-OFF in winter
If T3 > T8 + 5 then turn D ON until T3 < T8 +1, then turn D OFF
Runaway Conditions
If T3 > 200°F then turn D OFF
Fail Safe Conditions:
If communication is lost, FAIL OFF

Conditions Affecting Pump E:
If pump A OR pump D is ON, the turn pump E ON
If pump A AND pump D is OFF, then turn pump E OFF
Fail Safe Conditions
If communication is lost, FAIL OFF

Though this installation is using a 6 zone switching relay box manufactured by TACO, future installations may use ModBus instead.

A flow chart or wiring schematic, such as the one depicted above, shows start & termination points, processing, input & output functions, decisions and loops. Flow charts are among the oldest schematics for software.

What most people perceive as “programming”, the act of writing code in a programming language, only begins after the above analysis has been completed. Many programming languages exist, and what they all have in common is that they have a strict and rigid “grammar” (called syntax) and the ability to invent new “words” from a very small core vocabulary. The biggest difference between programming languages and natural languages, however, is that programming languages were devised to allow communication with a machine. A machine, or another programmable device, does exactly as instructed —no more, no less. A machine does not assume anything about its environment; it neither anticipates any future instructions, nor remembers what it did a fraction of a second ago. The “small, discrete, deterministic steps” that the preceding paragraphs mentioned must, hence, also be precise and comprehensive.

Flow charts, or other kinds of charts, can still be made with pencil on paper, but building a program that a computer can run requires special tools —tools with names like compiler, linker/locator, editor and debugger. Most programmers are also “power users” on their computers and the tools that they use are frequently less polished than your favorite office suite. On the other hand, the tools made for programmers often focus on letting you work efficiently, rather than wasting your time with animations, silly sounds and other attempts to look cool. Briefly: be not deceived by the Spartan interface of programmer’s tools.

Scripting programs for HVAC systems, even elaborate integration of solar thermal, solar PV, GSHPs, desuperheaters, thermal battery systems, and energy recovery ventilators (ERVs), etc., are relatively simple and straightforward to conceive, even for multiple stage heating, cooling, domestic hot water, and ERV systems.

Integrated Controls, LLC is developing a library of scripts for the eZEio Integrated Control Systems. Each of these integrated controls is being tested and proven in the field for use by HVAC, solar thermal, solar PV, thermal battery, and ERV installers.

PAWN Scripting for eZEio Integrated Controller

[Bolded blue text in brackets and yellow highlighted text is inserted for instructional purposes only, it is not part of the script]

[Input Definitions]

```
#define IN_T1          1
#define IN_T2          2
#define IN_T3          3
#define IN_T8          8
#define IN_Y           14

#define IN_LOGA            9
#define IN_LOGB            10
#define IN_LOGC            11
#define IN_LOGD            12
#define IN_LOGE            13
```

[Output Definitions]

```
#define OUT_A          1
#define OUT_B          2
#define OUT_C          3
#define OUT_D          4
#define OUT_E          5

#define OUT_A_SUMMER       7       [switch for summer status]
#define OUT_A_WINTER       8       [switch for winter status]
```

[Operational Definitions]

```
#define CHARGEDIFF     50          [ΔT 5°F turn ON]
#define CHARGEDIFFOFF      10      [ΔT 1°F turn OFF]
```

```
#define SOLARCHARGE             1200        [solar DHW preheat tank >120°F]
#define SOLARCHARGEOFF          900         [solar DHW preheat tank <90°F]

#define BATTTFER        700                 [>70°F transfer tank threshold]
#define BATTTFEROFF             660         [<66°F transfer tank threshold]

#define GROUNDLOOPLOW           420         [<42°F winter low threshold, turn ON ground loop]
#define GROUNDLOOPLOWOFF        450         [>45°F winter high threshold, turn OFF ground loop]
#define GROUNDLOOPHIGH          700         [>70°F summer high threshold, turn ON ground loop]
#define GROUNDLOOPHIGHOFF       660         [<66°F summer low threshold, turn OFF ground loop]

#define OVERTEMPSOLAR           2000        [>200°F over-ride/shut OFF solar CP]
#define OVERTEMPTRANSFERTANK            1000    [<100°F over-ride/shut OFF transfer tank CP]

// Month, Day                               [calendar – seasonal parameters for specific locality]
new const SEASONS[] = {
  4, 15,  // Summer start
  9, 15,  // Summer end
  10, 1,  // Winter start
  2, 18             // Winter end
};


new T1, T2, T3, T8;
new tick = 0;

main()                                      [Start-up Script]
{
  // This code will run on startup only
  PDebug("Hello world!");
  SetOutput(OUT_A_SUMMER, 1);
  SetOutput(OUT_A_WINTER, 1);
}


// returns 1 for summer, 2 for winter or 0 for fall/spring
findSeason()
{
   if((GetMonth() > SEASONS[0]) || (GetMonth() == SEASONS[0] && GetDay() >= SEASONS[1]))
          if((GetMonth() < SEASONS[2]) || (GetMonth() == SEASONS[2] && GetDay() < SEASONS[3]))
             return 1; // Summer!

   if((GetMonth() > SEASONS[4]) || (GetMonth() == SEASONS[4] && GetDay() >= SEASONS[5]))
          return 2; // Winter!
   if((GetMonth() < SEASONS[6]) || (GetMonth() == SEASONS[6] && GetDay() < SEASONS[7]))
      return 2; // Winter

   return 0;
}
```

```
mlTempF(inputno)
{
	new x = GetInputValue(inputno) * 1125;
	return (x-670000)/1000;
}

processA()                                          [Associated with Conditions for CPA]
{
	static c1 = 0;
	static c2 = 0;
	new x = 1;

	if( T3 > (T1+CHARGEDIFF) )		c1 = 1;
	if( T3 < (T1+CHARGEDIFFOFF) )		c1 = 0;

	if( T8 > SOLARCHARGE )			c2 = 1;
	if( T8 < SOLARCHARGEOFF )		c2 = 0;

	if(GetOutputState(OUT_A_WINTER)) {		// Winter switch is ON
		if(GetOutputState(OUT_A_SUMMER)) {	// Summer switch is ON
			if( c1==0 || c2==0 )
				x = 0;
		}
		else {					// Summer switch is OFF
			if( c1==0 )
				x = 0;
		}
	}
	else {						// Winter switch is OFF
		if(GetOutputState(OUT_A_SUMMER))// ..summer is ON
			x = 0;				// Never activate
	}

	if( T3 > OVERTEMPSOLAR )
		x = 0;
	if( T1 > OVERTEMPTRANSFERTANK )
		x = 0;

	SetOutput(OUT_A, x);
}

processB()                                          [Associated with Conditions for CPB]
{
	static c1 = 0;
	static c2 = 0;

	if( T1 > (T2+CHARGEDIFF) )  c1 = 1;
	if( T1 < (T2+CHARGEDIFFOFF) )	c1 = 0;

	if( T2 > BATTTFER )		c2 = 1;
	if( T2 < BATTTFEROFF )		c2 = 0;
```

```
	if( GetInputState(IN_Y) || c1 || c2 )
		SetOutput(OUT_B, 0); // Blue light ON
	else
		SetOutput(OUT_B, 1); // Blue light OFF

}

processC()                                          [Associated with Conditions for CPC]
{
	static c1 = 0;
	static c2 = 0;

	if( T1 < GROUNDLOOPLOW )          c1 = 1;
	if( T1 > GROUNDLOOPLOWOFF )       c1 = 0;

	if( T1 > GROUNDLOOPHIGH )         c2 = 1;
	if( T1 < GROUNDLOOPHIGHOFF )      c2 = 0;

	if( c1 || c2 )
		SetOutput(OUT_C, 0);
	else
		SetOutput(OUT_C, 1);
}

processD()                                          [Associated with Conditions for CPD]
{
	static c1 = 0;

	if( T3 > (T8+CHARGEDIFF) )  c1 = 1;
	if( T3 < (T8+CHARGEDIFFOFF) )       c1 = 0;

	if(GetOutputState(OUT_A_WINTER))            // Winter switch is ON
		if(GetOutputState(OUT_A_SUMMER) == 0)     // ..and summer is OFF
			c1 = 0;                           // Always turn off D

	if( T3 > OVERTEMPSOLAR )
		c1 = 0;

	SetOutput(OUT_D, c1);
}

processE()                                          [Associated with Conditions for CPE]
{
	new x = 0;

	if( GetOutputState(OUT_A) )
		x = 1;
	if( GetOutputState(OUT_D) )
		x = 1;
```

```
        SetOutput(OUT_E, x);
}

@Tick(uptime)                                   [General System Programming]
{
  // This event is called once per second
  // uptime is the number of seconds since last system reset

        T1 = mlTempF(IN_T1);
        T2 = mlTempF(IN_T2);
        T3 = mlTempF(IN_T3);
        T8 = mlTempF(IN_T8);

        if( GetOutputState(OUT_A_SUMMER) || GetOutputState(OUT_A_WINTER) ) {

                if(--tick <= 0) {
                        new s = findSeason();
                        if(s == 1) {
                                SetOutput(OUT_A_SUMMER, 1);
                                SetOutput(OUT_A_WINTER, 0);
                        }
                        else
                        if(s == 2) {
                                SetOutput(OUT_A_SUMMER, 0);
                                SetOutput(OUT_A_WINTER, 1);
                        }
                        else {
                                SetOutput(OUT_A_SUMMER, 1);
                                SetOutput(OUT_A_WINTER, 1);
                        }
                        tick = 600;             // Check season every 10 minutes
                }

                processA();
                processB();
                processC();
                processD();
                processE();
        }

        SetInputValue(IN_LOGA, GetOutputState(OUT_A));
        SetInputValue(IN_LOGB, GetOutputState(OUT_B));
        SetInputValue(IN_LOGC, GetOutputState(OUT_C));
        SetInputValue(IN_LOGD, GetOutputState(OUT_D));
        SetInputValue(IN_LOGE, GetOutputState(OUT_E)); }

@Alarm(sourcetype, sourceno, alarmno)           [System Alarm Notifications]
{
  // Any alarm will trigger this event
  // sourcetype is one of SOURCE_INPUT, SOURCE_SCHEDULE, SOURCE_TIMER
```

```
  // sourceno is the input/schedule/timer number
  // alarmno is the alarm number (1-4)
}

@Restore(sourcetype, sourceno, alarmno)
{
  // Any restore will trigger this event
  // sourcetype is one of SOURCE_INPUT, SOURCE_SCHEDULE, SOURCE_TIMER
  // sourceno is the input/schedule/timer number
  // alarmno is the alarm number (1-4)
}

@Timer(timerno)
{
  // called when a timer set by SetTimer triggers }

@ModbusReply(address, command, length, data[]) {
  // called as a result of ModbusSend when a modbus device sends a reply
  // address is the device modbus address
  // command is the modbus command
  // length is the number of bytes received (minus address, command and checksum)
  // data is the payload data, starting with the byte after the command byte }
```

Connections & Installation

Things to consider before installing the eZEio™ controller. The ezeio™ controller is designed for indoor use and should be installed in a dry and clean location. Do not expose the controller to rain or water, and avoid extreme temperatures. See the technical specifications for acceptable ranges.

The ezeio™ controller is a low voltage device. Never connect high voltage to the inputs or outputs, and only use the supplied AC adapter to power the controller.

Do not run wires that connects to the ezeio™ (Inputs / outputs / MicroLAN /Ethernet / ModBus / power or antenna) together with high voltage wiring. Use separate conduits whenever possible, and avoid environments with excessive RF or magnetic radiation as this may interfere or even destroy the

eZEio™ controller.

Take necessary precautions to avoid large static discharges to the eZEio™ connections.

WARNING ABOUT MICROLAN AND PHONE CONNECTORS

Never use connectors or wires designed for phone networks for connecting MicroLAN devices. Phone connectors usually alter the polarity, and will permanently damage MicroLAN devices, voiding warranty.

Testing & Troubleshooting

Once we have gone through it all —when the analysis of the requirements and the decomposition into small steps lie behind us and after having keyed in the program code, we can run the program and watch its output. Often, the program does not behave as intended right away. It may not even run in its very first version. This may be due to simple typing errors, which are easily fixed; but it may also be due to the programmer not properly understanding the problem. Naturally, we now have to find and fix the errors in the program, which is a cycle very similar to the one for the initial development:

- understand the program's behavior and what the program should be doing instead
- find a solution/fix in small discrete steps,
- write these steps in a programming language,
- and review and test the program.